Description

COMMON NAME SPACE FOR LONG AND SHORT FILENAMES

Technical Field

The present invention relates generally to data processing systems and, more particularly, to a common name space for long and short filenames.

Background of the Invention

Many operating systems, such as the MS-DOS, version 5, operating system, sold by Microsoft Corporation of Redmond, Washington, support only short filenames. In the MS-DOS, version 5, operating system, filenames may be a maximum length of eleven characters. Each filename may have a main portion of eight characters followed by an extension of three characters. An example filename in the MS-DOS, version 5, operating system is "EXAMPLE1.EXE", wherein "EXAMPLE1" constitutes the main portion and "EXE" constitutes the extension.

Each filename is limited to eleven characters due to constraints in the file system of the MS-DOS, version 5, operating system. This file system employs a directory structure in which each file has a directory entry associated with it. Unfortunately, the directory entry for a file only supports filenames with a maximum length of eleven characters. Such a limit in the length of the filenames is often frustrating to a user. The length limit of eleven characters prevents a user from employing adequately descriptive filenames and, in many instances, forces a user to insert awkward abbreviations of descriptive terms into the filename.

Summary of the Invention

It is, therefore, an object of the present invention to provide a system that supports long filenames.

It is another object of the present invention to provide a system that supports long filenames while minimizing the compatibility impact of supporting long filenames.

It is a further object of the present invention to provide a system that supports a common name space for both long filenames and short filenames.

In accordance with the first aspect of the present invention, a method is practiced in a data processing system having a memory means and a processing means. In accordance with this method, a first directory entry is created and stored in the memory means for a file. The first directory entry holds a first filename for the file and information about the file. A second directory entry is also created and stored in the memory means. The second directory entry holds at least one portion of a second filename having a fixed number of characters and information about the file. One of the first or second directory entries is accessed in the memory means to gain access to the information contained therein.

In accordance with another aspect of the present invention, a data processing system includes a memory that holds a first directory entry for a file, a second directory entry for the file, and an operating system. The first directory entry includes a first filename for the file and the second directory entry includes the second filename for the file. The second filename includes more characters than the short filename. The data processing system also includes a processor for running the operating system and accessing either the first directory entry or the second directory entry to locate the file.

3

In accordance with yet another aspect of the present invention, a method is practiced in a data processing system having memory. In accordance with this method, a file is created and the file is assigned a user-specified long filename. The long filename is manipulated with the data processing system to create a short filename of fewer characters. The long filename and the short filename are stored in memory so that the file can be accessed by either the long filename or the short filename.

In accordance with a further aspect of the present invention, a method is practiced in which a first directory entry for a file is stored in a memory means. The first directory entry holds the short filename for the file. The short filename includes at most a maximum number of characters that is permissible by an application program. A second directory entry is also stored in the memory means for the file. A second directory entry holds at least the first portion of a long filename for the file. The long filename includes a greater number of characters than the maximum number of characters that is permissible by the application program. The application program is run on a processor of the data processing system. The application program identifies the file by the short filename.

In accordance with a still further aspect of the present invention, a method is practiced in which a first directory entry is stored in the memory means for a file. The first directory entry holds a short filename for the file that includes at most the maximum number of characters that is permissible by the operating system. A second directory entry is stored in the memory means for the file. The second directory entry holds a long filename for the file that includes more than the maximum number of characters that is permissible by the operating system. In this instance, the operating system does not use long filenames; rather, it uses solely short

filenames. The first directory entry is accessed by the operating system to locate the file.

Brief Description of the Drawings

A preferred embodiment of the present invention will now be described herein with reference to the Drawings. The Drawings include the following Figures.

Figure 1 is a block diagram of a data processing system used for implementing the preferred embodiment of the present invention.

Figure 2 is a block diagram illustrating the storage of short filename directories in locations adjacent to long filename directory entries.

Figure 3a shows the format of a short filename directory entry in the preferred embodiment of the present invention.

Figure 3b shows the format of a long filename directory entry in the preferred embodiment of the present invention.

Figure 4 is a flow chart illustrating the steps performed by the preferred embodiment of the present invention when a new file is created.

Figure 5a is a flow chart illustrating the steps performed in creating a long filename directory entry in the preferred embodiment of the present invention.

Figure 5b is a block diagram illustrating bits in the file attributes fields of the long filename directory entry of Figure 3b.

Figure 6a is a flow chart illustrating the steps performed when a short filename is provided by the user in the preferred embodiment of the present invention.

Figure 6b is a flow chart illustrating the steps performed when a long filename is provided by user in the preferred embodiment of the present invention.

Detailed Description of the Invention

A preferred embodiment of the present invention described herein provides support for the use of long filenames (i.e., filenames that may have substantially more characters than current operating systems, such as the MS-DOS, version 5, operating system permit). "Short filenames" will be used hereinafter to refer to filenames that have a small limit (such as 11 characters) as to the maximum number of characters permitted. In the preferred embodiment, the long filenames are provided in a common name space with the short filenames. A long filename and a short filename are provided for each file in the system. The sharing of a common name space is realized through providing separate directory entries for long filenames and short filenames. Each file has a short filename directory entry associated with it and may also have at least one long filename directory entry. The short filenames are like those provided previously in the MS-DOS, version 5, operating system. The long filenames, as will be described in more detail below, may have a maximum length of up to 255 characters. The preferred embodiment will be described with reference to an implementation with the MS-DOS, version 5, operating system.

The potential compatibility problems of supporting long filenames are apparent by considering one solution to the problem of short filenames. This solution is not part of the present invention and is described herein merely to illustrate how a preferred embodiment avoids the compatibility problems suffered by this proposed solution. This solution supports long filenames by merely increasing the number of characters the operating system permits for a filename.

There are two major difficulties with this solution. First, the existing application bases of many systems use only short filenames (e.g., 11 characters or less) and are not prepared to utilize only long filenames (e.g., up to 255 characters). As an example, an

application may allocate a buffer large enough to hold the short filename and if the operating system tries to place long filename data into this buffer, the buffer may overflow so as to cause the application data to be unexpectedly overwritten. Second, certain disk utility programs access the file system volume directly and, thus, do not rely on the operating system to access the files. If the file system is changed to support long filenames, compatibility problems with the disk utility programs arise.

The preferred embodiment of the present invention described herein, in contrast, seeks to minimize the compatibility impact of supporting long filenames by providing both a long filename and a short filename for each file. As a result, applications and utilities that require short filenames still have short filenames available, and applications that use long filenames have long filenames available.

The preferred embodiment of the present invention may be implemented as code realized as software or firmware. In order to support long filenames, the preferred embodiment of the present invention provides several long filename application program interfaces (APIs). These APIs are provided along with the conventional short filename interfaces that are standard with the operating system. The long filename APIs support file operations and directory entries for long filenames. The APIs include a file attributes function, a file delete function, a file directory function, a file find function, a file open/create function and a file rename function.

The preferred embodiment of the present invention may be implemented in a data processing system 10 like that shown in Figure 1. This data processing system 10 includes a central processing unit (CPU) 12 with a standard set of registers 13 that includes an accumulator (AL) register 15, a memory 16 and input/output (I/O) devices 14. The CPU 12 oversees the

7

operations of the data processing system 10 and has access to the memory 16 and the I/O devices 14. The memory 16 may include both RAM and disc storage. The memory 16 holds an operating system 17 (denoted as "O.S." in Figure 1) which includes the long and short filename APIs. Those skilled in the art will appreciate that the present invention may be implemented on other suitable data processing systems.

All of the functions for the long filename APIs and short filename APIs are incorporated into the operating system 17. Those functions are supported through an Int 21h interrupt call (where 21h denotes 21 in hexadecimal notation). In other words, all the functions are called by executing an Int 21h interrupt, wherein the function that is called through the Int 21h interrupt is specified by a value placed in a register, as will be described in more detail below. The Int 21h interface is like that provided in the MS-DOS, version 5, operating system except that the interface also supports calls to functions for long filenames. In calls to the long filename APIs, the function number to be called is placed in the AL register 15 of a processor, such as the CPU 12 in Figure 1 before the interrupt is initiated.

In order to support both a long filename and a short filename for each file, the preferred embodiment provides a short filename directory entry 18 (Figure 2) and may provide at least one long filename directory entry 20 for each file in a common name space. Each file has a long filename and a short filename associated with it. A long filename directory entry 20 is only created when the long filename cannot be correctly stored in the short filename directory entry. The long filename directory entries 20 are stored adjacent to the corresponding short filename directory entry 18 as part of the common name space used in memory 16. Moreover, the long filename directory entries 20 are configured to

minimize compatibility problems with operating systems that support only short filenames.

Figure 2 shows an example of the directory entries 18 and 20 for a file in the preferred embodiment described herein. The short filename directory entry 18 is provided along with several long filename directory entries 20. The number of long filename directory entries 20 provided (including zero long filename directory entries) for a file depends upon the number and type of characters in the long filename. As will be described in more detail below, each long filename directory entry 20 may hold up to 26 characters of a long filename. The long filename directory entries 20 are dynamically allocated based upon the number of characters in the long filename. For example, a file with a long filename of 50 characters has two long filename directory entries 20 allocated for it, whereas a file with a long filename of 70 characters has three long filename directory entries 20 allocated for it. As was mentioned above, a long filename may have a maximum of 255 characters and thus, a maximum of 10 long filename directory entries 20 may be allocated for any file. The maximum of 255 characters per filename is a product of maximum path length (260 characters) limitations of the operating system 17.

There may be many instances in which the long filename does not completely fill all of the space available in the allocated long filename directory entries 20. In such an instance, a null terminator is placed after the last character of the long filename so that additional spaces or nonsensical data will not be returned. The extra spaces are filled with 0FFh (where "h" indicates the use of hexadecimal notation).

Figure 3a illustrates the format of the short filename directory entry 18. Each of the fields in the directory entry begins at a different offset relative to the starting address of the directory entry. A filename

9

field 22 holds the main portion (i.e., the leading 8 characters) of the short filename. As the main portion of the short filename may hold up to eight characters of the short filename, the filename field 22 is eight bytes in length and begins at offset 00h. The filename field 22 is followed by a file extension field 24 at offset 08h. The file extension field holds the characters of the extension of the short filename. The extension field 24 is three bytes in length (encoding three characters).

Following the extension field 24 at offset 0Bh is a file attributes field 26. The file attributes field 26 includes a number of bits that, based upon whether the bits are set or not, specify information about the associated file.

The short filename directory entry 18 also includes a reserved field 28. The reserved field 28 begins at offset 0Ch and is ten bytes in length. The short filename directory entry 18 additionally includes a time of last update field 30 and a date of last update field 32. The time of last update field 30 is two bytes in length and begins at offset 16h. The date of last update field 32 is two bytes in length and begins at offset 18h.

The short filename directory entry 18 includes a beginning disk cluster field 34. The beginning disk cluster field 34 holds a pointer to the section of the memory 16 (Figure 1) where the file's first disk cluster is held (i.e. to the beginning of the allocation chain for the file). This beginning disk cluster field 34 (Figure 3a) is stored at offset 1Ah and is two bytes in length. A file size field 36 follows the beginning disk cluster field 34. The file size field 36 holds a value that specifies the amount of memory occupied by the file associated with the short filename directory entry 18. The file size field 36 is four bytes in length and begins at offset 1Ch.

10

Figure 3b illustrates the format used for each of the long filename directory entries 20. The long filename directory entry 20 additionally includes a signature field 38 that holds a digital signature. The signature field 38 is useful in specifying the order of a long filename directory entry 20 in a sequence of associated long filename directory entries. For example, a first long filename directory entry includes a signature field 38 that specifies that it is the first entry, and each successive long filename directory entry includes a signature field 38 that specifies where the long filename directory entry fits in the sequence of long filename directory entries for a file. The signature field 38 is provided primarily for use with utility programs that directly access the file system volume. The signature field 38 is one byte in length and begins at offset 00h, which is the beginning of the filename field 22 (Figure 3a) of the short filename directory entry 18. The signature field 38, given its location in the long filename directory entry, might easily be mistaken for a portion of a short filename by certain utility programs. Hence, the signature field 38 includes only illegal short filename characters so that the characters may not be readily changed by systems or utilities that support only short filenames.

The long filename directory entry 20 includes three fields 40, 48 and 52 that are provided for holding characters of the long filename. The first long filename field 40 begins at offset 01h and may hold up to ten characters of the long filename (i.e., it is 10 bytes in length). The second long filename field 48 begins at offset 0Eh and may hold up to twelve characters (i.e., 12 bytes) of the long filename. Lastly, the third long filename field 52 begins at offset 1Ch and may hold up to four characters (i.e., 4 bytes) of the long filename. Thus, cumulatively, these three fields 40, 48 and 52 may hold up to twenty-six characters of the long filename.

The long filename fields 40, 48 and 52 are filled sequentially beginning with field 40 and then filling fields 48 and 52, consecutively.

While the long filename directory entry 20 differs from the short filename directory entry 18, the long filename directory entry 20, nevertheless, includes certain similar fields at the same specified offsets as were discussed above for the short filename directory entry 18 (Figure 3a). As such, operating systems that do not support long filenames are not disturbed by the long filename directory entries 20. For instance, the long filename directory entry 20 includes a file attributes field 42 which is like the file attributes field 26 (see Figure 3a) provided in the short filename directory entry.

The long filename directory entry 20 contains a checksum field 44, which is one byte in length and at offset 0Dh. The checksum field 44 holds a checksum of the short filename. The checksum byte, as will be described in more detail below, is used to ensure that the long name is valid for the associated short filename and to act as a pointer to the short filename directory entry 18 that is helpful to disk utility programs. A flags field 43 is held at offset 0Ch. The flags field 43 holds a flag bit that may be set when unicode characters are used.

In addition, the beginning disk cluster field 50 (Figure 3b) of the long filename directory entry 20 is analogous to the beginning disk cluster field 34 (Figure 3a) of the short filename directory entry 18. However, it always has a constant value of zero in the long filename directory entry.

The above discussion has focused on how the directory entries 18 and 20 (Figure 2) are used to support both long filenames and short filenames. The discussion below will focus on how such directory entries are supported by the preferred embodiment of the present invention.

12

When a new file is created, the preferred embodiment must take steps to support both a long filename and a short filename for the new file. In discussing how the preferred embodiment supports both long filenames and short filenames, it is helpful to first focus on the creation of the directory entries and then to focus on the creation of the filenames. Figure 4 is a flowchart depicting the basic steps performed upon creation of the new file. Initially, the new file is created (step 54) using either a long filename API or a short filename API. Both varieties of APIs support the creation of files. Depending on the type of API that is used to create the files, the file will initially have a long filename and/or a short filename. In other words, if a file is created with a long filename API, it will initially have a long filename and if a file is created with a short filename API, it will initially have a short filename, which may also be the long filename for the file.

At least one long filename directory entry 20 may be created for the file. First, a determination is made whether a long filename directory entry 20 is required (step 51). If the long filename will not correctly fit in the short filename directory entry 18, a long filename directory entry 20 is required. Long filename directory entries 20 are dynamically allocated based upon the number of characters in the long filename. At a minimum, a short filename directory entry 18 will be created that has the format that is shown in Figure 3a. Thus, the system checks to see how many long filename directory entries are needed and allocates space for the short filename directory entry and as many additional long filename directory entries as are required (step 58). It should be appreciated that when both a short filename directory entry 18 and at least one long filename directory entry 20 are created, space for both types of directory entries are allocated together at the same time. The long and short filename directory entries 18 and 20

are then filled in step 59. However, if no long filename directory entry is required, no space will be allocated (i.e., steps 58 and 59 are skipped).

Figure 5a is a flowchart depicting the steps performed in filling in a long filename directory entry 20 (see step 59 in figure 4). The steps are shown in a given sequence, but those skilled in the art will appreciate that the steps need not be performed in this illustrated sequence. Rather, other sequences are equally acceptable.

A hidden bit in the file attributes field 42 is set to have a value of one (step 62). Figure 5b shows the bits included in the file attributes field 42. The hidden bit is designated by the letter "H" in Figure 5b and is present at bit position 1 in the file attributes field 42. When the hidden bit is set to a value of one, the directory entry is hidden and is excluded from normal searches of the directory entries 18 and 20. By setting the hidden bit, the long filename directory entries 20 (Figure 2) are not searched in conventional directory entry searches. The hidden bit is set so that down level systems (i.e., systems that support only short filenames) will not see the long filename directory entries 20.

A read-only bit is also set in the file attributes field (step 64 in Figure 5a). The read-only bit is designated by the letter "R" in Figure 5b and is present at bit position 0 in the file attributes field 42. Setting the read-only bit to a value of one indicates that the file is a read-only file and any attempts to write to the file will fail.

A system bit in the file attributes field 42 is set to a value of one (step 66 in Figure 5a). The system bit is designated by the letter "S" in Figure 5b and is present at bit position 2 in the file attributes field 42. Setting the system bit to a value of one designates the file as a system file and excludes the directory entry from normal searches of the directory entries 18 and 20. The setting of the system bit to a value of one hides the

long filename directory entries 20 from down level operating systems that support only short filenames.

Next, a volume label bit is set in the file attributes field 42 (step 68 in Figure 5a). The volume label bit is designated by the letter "V" in Figure 5b and is present at bit position 3 in the file attributes field 42. Setting the volume label bit to a value of one hides the long filename directory entry from "Check Disk" operations of certain disk utility programs. For example, MS-DOS, version 5.0, includes a utility named CHKDSK. The setting of the volume label attribute hides the long filename directory entries from CHKDSK.

The discussion will now return again to the flowchart of Figure 5a. The signature byte field 38 (Figure 3b) is filled with a digital signature (step 70 in Figure 5a). As was mentioned above, the signature distinguishes the order of the long filename directory entries 20 for the file. The checksum field 44 in Figure 3b is filled with the appropriate checksum of the short filename (step 72 in Figure 5a). The checksum byte field 44 (Figure 3b) is used to associate the long filename directory entries 20 with their appropriate short filename by holding a checksum of the short filename. The beginning disk cluster field 50 (Figure 3b) is set to zero (step 74 in Figure 5a). The long filename directory entry 20, thus, has no data allocated to it. This helps to make the long filename directory entry invisible in down level systems. Lastly, the bits for the characters of the long filename are stored in the appropriate long filename fields 40, 48 and 52 (Figure 3b) of the long filename directory entry 20 (step 76 in Figure 5a).

By setting the file attributes field 42 (Figure 5b) bits as described above and by setting the beginning disk cluster field 50 to zero (Figure 3b), the preferred embodiment of the present invention makes the long filename directory entries nearly invisible to operating systems that support only short filenames (i.e., down

level systems). Nevertheless, files with long filenames are still permitted in down level operating systems. The long filename directory entries are not visible in the directory entry listing for down level systems. The combination of bit settings in the file attributes field and the zeroing of the beginning disk cluster field 50 make the long filename directory entries invisible to down level systems. Thus, compatibility problems arising from having long filenames in the down level operating system are minimized. Moreover, utility programs, that may skew the order of directory entries, are not a problem. The signature field 40 (Figure 3b) and the checksum field 44 may be used in conjunction to rearrange entries that are out of order. In particular, the checksum fields 44 are used to associate long filename directory entries 20 with a short filename directory entry and the signature fields 40 of the long filename directory entries are used to assign related long filename directory entries into proper sequence.

The discussion above has noted that filenames are created using either short filename APIs or long filename APIs. As a result, when a file is created it has either a long filename or short filename assigned to it by the user, depending on whether a long filename API or short filename API is used. The preferred embodiment of the present invention described herein automatically creates the missing short filename or long filename. For instance, if a file is created using a short filename API, the preferred embodiment described herein establishes a corresponding long filename (which is the same as the short filename). Analogously, if a file is created using a long filename API, the preferred embodiment generates a corresponding short filename that is stored in a short filename directory entry 18. Figure 6a shows the steps performed by the preferred embodiment when the short filename is provided by the user. In particular, the user provides a short filename (step 78 in Figure 6a), and the

16

short filename is used as the long filename (step 80). When the user provides a short filename, the system checks whether the name is a valid short filename and whether there are any existing files that pose a conflict (not shown). If there is no problem in terms of format or conflict, the file is assigned the provided short filename. The short filename is then used as the long filename, and there is no long filename directory entry for the file.

When a file is created using a long filename API, the resulting creation of a corresponding short filename may be quite complex. Figure 6b is a flowchart illustrating the steps performed to create the short filename in such an instance. Initially, the long filename is provided by the user (step 82 in Figure 6b). The preferred embodiment then checks whether the long filename is a valid short filename (step 84). If the long filename is a valid short filename, the long filename is used as the short filename (step 86).

However, if the long filename does not qualify as a valid short filename, a short filename is created by removing the spaces from the long filename and using the resulting characters as a proposed short filename (step 88). Initial periods, trailing periods and extra periods that are prior to the last embedded period are then removed from the proposed short filename (step 90). Furthermore, any illegal short filename character is translated into an underscore (step 92). A check of whether the proposed short filename contains an extension is then performed (step 94). If the proposed short filename contains an extension, the leading main portion of the filename is truncated to six characters in length, and the leading three characters of the extension are used (step 96). Subsequently, a "~1" is appended to the leading portion of the remaining characters (step 98) to serve as the short filename.

If the modified long filename does not contain an extension (step 94), the long filename is truncated to six characters (step 100), and "~1" is appended to the truncated filename (step 102) to serve as the short filename. In both of the above-described instances (i.e., the "yes" instance and "no" instance of step 94), the preferred embodiment next checks whether the proposed short filename collides with any other short filename (step 104). If the proposed short filename does not collide with another short filename (i.e., there is no other identical short filename), the proposed short filename is assigned as the short filename for the file (step 112). In the case where the proposed short filename collides with another short filename, the characters that are appended to the name are incremented by one (step 106). Thus, if the number value is initially "~1", the number value is incremented in step 106 by one to "~2". The preferred embodiment checks whether the new proposed short filename exceeds eight characters in length (step 108). If the new proposed short filename does not exceed eight characters in length, the checking of whether the proposed short filename collides with another short filename is repeated (step 104). When the number of characters in the filename exceeds eight characters in length, the new short filename is shortened to eight characters (step 110). In particular, if the length of the leading portion of the filename (ignoring the extension) plus the tilda and the number exceeds eight characters, the leading portion of the filename is shortened until the new proposed short filename (absent the extension) fits in eight characters. For example, the filename "MonKey~10.EXE" is shortened to "MonKe~10.EXE." The above-described steps 104, 106, 108 and 110 are repeated until a short filename is created for the file that is of proper length and that does not collide with another short filename.

The preferred embodiment of the present invention provides a solution to the problem of short filenames while minimizing the compatibility impact of the solution. The use of a common name space that provides a long filename and a short filename for each file allows the files to be used both with applications that support short filenames and applications that support long filenames.

While the present invention has been described with reference to a preferred embodiment thereof, those skilled in the art will appreciate that various changes in scope and form may be made without departing from the present invention as defined in the appended claims.